SEF - Week 6 State diagrams

Introduction

Another diagram form is the **state diagram**, or **state chart**, or **finite state machine**. Instead of looking at how objects communicate with each other, we turn our attention to examining a single object, and how it responds to the messages it receives.

A state diagram examines the different states an object can be in, and how and when an object can transition from one state to another.
They are also known as finite state machines as they consist of a fixed set of possible states, and can be thought of as a mechanical machine in their descriptive simplicity. This simplicity lies at the heart of their attraction for many areas of computing - they are easily understood and proved correct.

What is a state?

A state is a condition, a particular set of values that an object holds for a time. A washing machine might be in the spin cycle state, or the filling state. A DVD player can be in the states paused/play/fast forward/backwards (all of which happen when it is in the on state).

A book in a library will be in the states "just purchased", "shelved", "borrowed", "overdue", "returned" and "disposed". The physical object is reflected in a computer system by a book object which may have an attribute which explicitly lists what state it is in (a string of characters, or a set of numbers (0 = "just purchased", 1 = "shelved" etc, or a more modern enumerated type (supported by most modern languages)).

The collection of states a library book can be in.

Alternatively its state may be (less explicitly) encoded by the associations between it and other objects. For example if there is a "currently borrowed" association between a library customer and a book, we know it is borrowed! We can determine if it is overdue by looking at the current date and the due date.

The system may also have a separate borrowed class which separately encodes the state of a borrowing.

Moving from one state to another

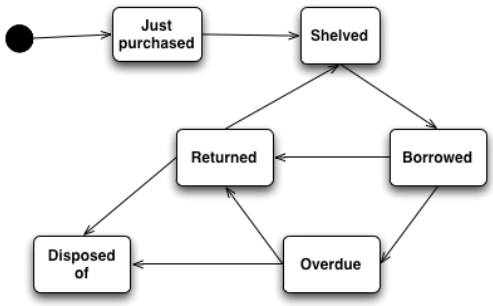
What causes these objects to transition from one state to another? Can you move from one state to any other state? What restrictions are there?

A DVD player moves between states when the user interacts with it by pressing buttons, but also when other ***events*** occur such as reaching the end of a movie, when it moves from the play state to the menu state.

A person being woken from a deep sleep by an alarm clock is likely to only slowly move from the sleeping state to a wakeful state - the period in between of befuddled grogginess could be modelled as a series of states (90% befuddled grogginess, 80% befuddled grogginess etc), but in reality state diagrams don't model systems that don't have crisp boundaries between states very well.

A separate field of study on fuzzy logic (see http://en.wikipedia.org/wiki/Fuzzy_control_system) caters for non discrete systems.

We show the possible ***transitions*** between states using a directed arc (arrow). The library book example from above is shown below.



Note the introduction of an initial state (a filled circle) which documents where the state diagram starts.

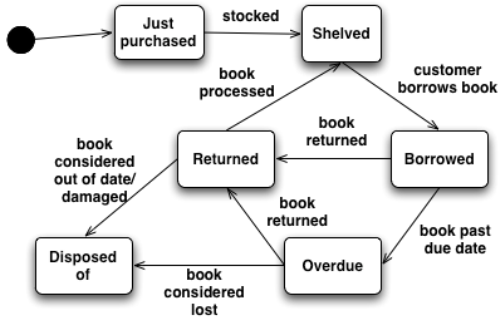
What is an event?

An event is an occurrence of an activity or a condition that becomes true. Pushing the eject button on a DVD player, or pressing the temperature button on a washing machine are all events. A recording device may start recording a TV show at 8pm; in this case the event is a particular time.

Both DVDs and washing machines are computing devices, so we need to consider how these external button pressing events get translated into computer system.
Through various means a button press is eventually transformed into a message send (also called a method/function call) (see the previous section on behavioural models) - the arrival of the message is the event. It may be an informative message ("the time is now 8pm") or it may be an update (change the spin level from gentle to normal). It could also be a message that doesn't result in an overall state change - an aircraft's flight control

system will have its altitude setting constantly updated as it moves, but it doesn't move into the "Terrain Alert" state until the altitude (and rate of descent) matches some specified values.

We annotate each arc in the state diagram with events to indicate when the object moves from one state to another. Note that a couple of the states have multiple arcs coming out of them. The events on the exiting arcs must not overlap in their semantics - otherwise the chosen path would be ambiguous.



How many states?

When designing a state diagram you need to ask yourself "How many states should I include?". You might be tempted to model all the possible states that an object can be in, however they are rather numerous.

Every variable in a computer can hold multiple states. A single bit has two states - 0 or 1. An 8 bit byte has 256, and a single 32 bit integer has over 4 billion states.
A moderately complex program is likely to have millions or possibly billions of 32 bit values yielding a phenomenally enormous number of states (the number of states is multiplied, not added, so you would have around 2^1,000,000,000 states).

Of course no one deals individually with this many states, instead we deal with abstractions of the possible states. Instead of viewing an integer as having over 4 billion states, we may be interested in only 3 of them - when the number is positive, 0, or negative.

In some cases there may be only a couple of pertinent attributes in an object that determine/document what state an object is in. A person's details may include their name, address and age. If we want to keep track of their eligibility for an old age pension then the state (eligible/ineligible state) only depends on their age. Their name and address are irrelevant to determining what state the person is in.

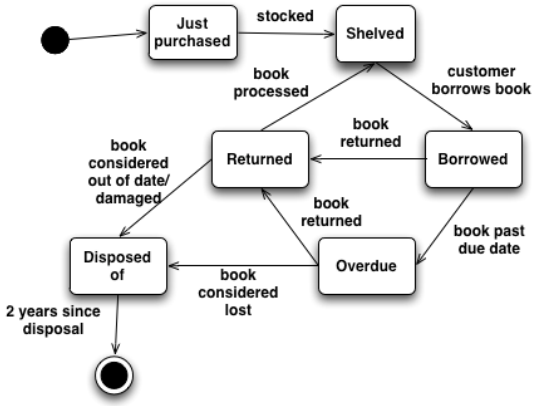
Final state

When an object is removed from our system we are no longer interested in what state it is in. We don't need to remember when a car's engine is due for service if it has been replaced. We can delete the car engine's details from our system. Similarly if someone turns the power off on a DVD player we don't need to remember if it was playing or in the main menu as when it is turned on it will revert to the main menu anyway*.

For this reason state diagrams can have a final "end" state - a bulls eye symbol - which indicates the state machine is no longer of any interest.

When a physical library book is disposed of or lost it can no longer be borrowed so many of the states are no longer accessible. You may want to keep the computer counterpart of the object as it may be linked to other objects for historical reasons (you want to have a history of people who have borrowed that copy) in which case there would be no final state.

An information retention policy that retains records for only two years could see the final state link to the "disposed of" state by a timing event.



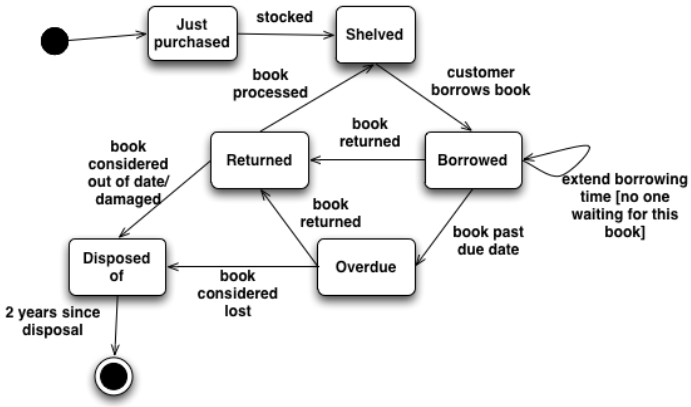
*but see history states below.

Guards on states

Some events are not always considered "open". A library book borrowing can be extended if no one else has requested it. Rather than building extremely elaborate state diagrams which include the state "someone waiting to borrow the book" a guard can be placed after the event to indicate whether that event is a possibility.
A guards must be written as a predicate (i.e. a statement that evaluates to true or false), enclosed in square brackets.

Predicates are sometimes confused with business rules, so the statement "Patients must be admitted by the receptionist" is a business rule, whereas

"Patient is admitted" is a query about the patients status and evaluates to true or false.



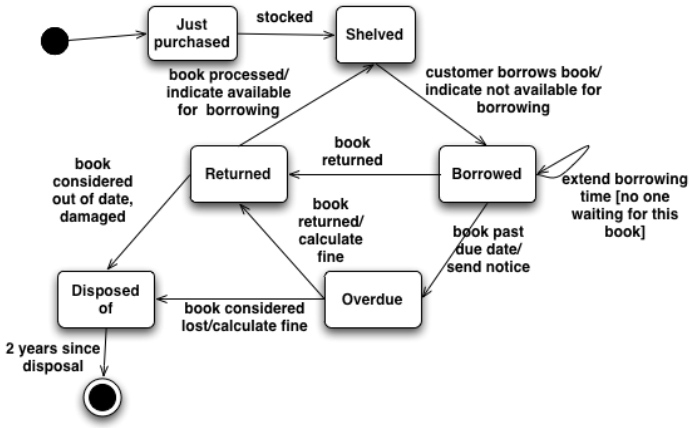
Actions

Whenever a transition is made an action can be triggered. This can affect the object internally, or it may result in a message being sent to another object in the system. For example if a plane flies below a threshold altitude it enters a state, and a triggering action is to get start the "Terrain Alert" system.
System can and have been modelled as sets of interacting state machines. Events trigger state changes in one object that trigger further actions, resulting in a cascade of messages possibly until the system settles down into a new (overall) stable state.
Flight control software continually deal with incoming events are never quiescent, but at a suitable level of abstraction we can still see some stable states such as flying, landing, and landed etc..

The syntax for actions is

event [guard] / action

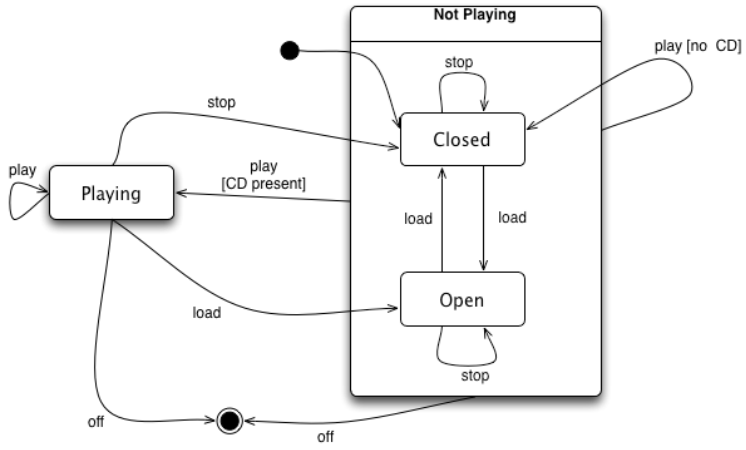
Multiple actions can be separated by commas.



Composite states

A DVD player has two states - Playing an Not Playing. When it is in the not playing state there are other states that it can also be in - tray open and tray closed. The Not Playing state is considered a composite state and is drawn with multiple states nested inside it.

You can transition from any outer state to a particular inner state (or vice versa) or you can traverse from the encompassing state ("Not Playing") to the Playing state.



In this diagram the events correspond to pressing the DVD's control buttons.

History states

Even when a state is left (especially a composite state) we may need to remember something about the system. On return to the composite state we could go back the state we left (for example).
A state that retains some history is labelled with a H in a circle.

Priestly discusses the need for history states.

Common mistakes in state diagram

A number of common mistakes made in state diagrams are listed below.

States that don't exist

A hospital system that has "Not yet admitted" as a state for a patient is irrelevant. Until the Patient has been seen for the first time we don't need to maintain any information about them.

Modelling the wrong things

A hospital patient that is discharged from a hospital doesn't disappear - they may return the next day. The state diagram should not go to a final (bulls eye) state on discharge as we still need to maintain information about that person (for at least another 7 years). The person would remain in a "discharged" state until their return.

Creating an activity diagram!

People new to state diagrams often find them hard to distinguish from an activity diagram.

If you find that the boxes contain actions rather than statenames you are doing it wrong.
If you can read it as a sequence of steps then it's possible that you are doing it wrong.

Note that this may be in the wording. "Admit" is a verb and therefore an activity. "Admitted" is an adjective - it describes an object and so is a reasonable name for a state.

Not nesting states

For a DVD state machine if you the "On" state is followed by the "Playing" state, does this mean the machine is not on when it is playing?
A machine can be both On and Playing at the same time, so this indicates the need for a composite state.